Software and Systems Modeling (2020) 19:1461–1463
https://doi.org/10.1007/s10270-020-00812-7

GUEST EDITORIAL

Editorial to theme section on interplay of model-driven and component-based software engineering

Federico Ciccozzi[1] · Antonio Cicchetti[1] · Andreas Wortmann[2]

Received: 8 June 2020 / Accepted: 10 June 2020 / Published online: 24 July 2020

This theme section aims to disseminate the latest trends in the use and combination of Model-Driven Engineering (MDE) and Component-Based Software Engineering (CBSE). On the one hand, MDE aims to increase productivity in the development of complex systems while reducing the time to market. On the other hand, CBSE aims to deliver and then support the exploitation of reusable "off-the-shelf" software components that can be incorporated into larger applications. An effective interplay of MDE and CBSE can yield benefits to both communities: the CBSE community would benefit from implementation and automation capabilities of MDE, the MDE community would benefit from the foundational nature of CBSE.

In total, we accepted three submissions for publication in the theme section after a thorough peer-reviewing process.

1 Model-driven engineering

MDE is an established methodology to increase the productivity of complex systems while reducing the time to market. It enables and suggests a shift from code-centric approaches to a more human-centric development, where models represent artefacts closer to human terminology and understanding. These models can be programmatically read and exploited for simplifying the design, implementation, execution, and evolution of software systems.

Models are defined with concepts that are less bound to the underlying implementation technology but closer to the problem domain, i.e. the concepts the modellers are familiar with. This makes the models easier to specify, understand, and maintain, which facilitates the understanding of complex problems and their potential solutions through abstraction.

By leveraging abstract models as primary development artefacts, MDE enables to systematically concentrate on different levels of abstractions, each providing a view for specific stakeholders, for instance (i) improving usability, (ii) enabling customizability in different and specific domains, (iii) promoting reusability of the different algorithms, methods, and techniques, (iv) managing variability and complexity both at design time and run-time, and (v) handling qualities like evolvability, changeability and configurability, modifiability, scalability, power consumption, and dependability. Models and model transformations are the core development artefacts in MDE.

In particular, MDE promotes shifting from source code specified in general-purpose programming languages to models expressed in explicit domain-specific modelling languages (DSMLs). DSMLs use metamodels or grammars to define the modelling concepts of a domain, as well as the relations between them, and different means to reify their semantics. A metamodel is an abstraction that highlights the characteristics of well-formed models, which are said to conform to their metamodel like a program conforms to the grammar of its programming language.

Through making modelling languages explicit, models can be subjected to automated analyses and syntheses, such as well-formedness checking or model transformation. Model transformations produce a non-empty set of target artefacts (i.e. models, text, binary files,…) from a non-empty set of source models. For example, by focusing on the software architecture domain, practitioners might take advantage of model transformations to automatically obtain program code, alternative model descriptions, deployment configurations, or inputs for analysis tools from their software architecture models.

An essential distinction of model transformations is that a model can be transformed either horizontally or vertically.

✉ Federico Ciccozzi
federico.ciccozzi@mdh.se

Antonio Cicchetti
antonio.cicchetti@mdh.se

[1] School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden

[2] Software Engineering, Department of Computer Science, RWTH Aachen University, Aachen, Germany

Horizontal transformation means that the source model is transformed into a model or another type of artefact at the same level of abstraction. Vertical transformation means that the source model is transformed into a model or another type of artefact at another level of abstraction. As the development of model transformations usually is a one-time effort, practitioners can easily (re)use model transformations defined by MDE tool providers and domain experts.

2 Component-based software engineering

CBSE initially emerged as a software discipline to deliver reusable "off-the-shelf" software components to be incorporated into larger applications. The main focus has been on effective and general-purpose reuse of components within a large variety of different applications. Nowadays, especially with the increasing development of Cyber-Physical Systems, the distributed manufacturing systems of Industry 4.0, and the ubiquitous presence of Internet of Things systems, CBSE continues to attract interest and evolve as a software approach and methodology for the efficient assembly of flexible software systems.

From the original design needs, mostly focused on promoting effective and efficient reuse of available third-party pieces of software, the attention of software engineers moved towards the definition of approaches and development of methods to add, remove, replace, modify, and assemble components dynamically, and during operation. For instance, in the domain of Cyber-Physical Systems, the strong connection between the computational and physical entities has been recognized, leading to the development of hybrid component frameworks. Such frameworks aim to be capable of taking into account and reason on both the event-based and discrete properties of computational entities and the time-based and continuous properties of physical entities.

From facing challenges introduced by the limitations of the previously leveraged object-oriented technologies, such as loose coupling, independent software reuse, seamless integration of heterogeneous software, etc. CBSE evolved, and indeed is still evolving, to address issues related to support the dynamicity, high interaction, and safety and dependability concerns of modern software systems. Very often, this led to rethinking widely adopted CBSE development processes to relax the traditional division among development phases by moving some activities from design time to deployment time and run-time. In these new and more dynamic development settings, the use of models at run-time (e.g. in the form of Digital Twins) has been found a key factor. In this direction, recent research focused on the definition of novel software development processes and methods to build highly dynamic and evolvable component-based systems.

The kind of systems targeted with CBSE ranges from component-based systems structured through component & connector styles to service-oriented and thing-based systems composed through either orchestration or choreography. For all of these, MDE technologies, including models at run-time, are deeply exploited to support analysis and automated synthesis methods for the production of the correct (concerning functional and extra-functional properties) integration of components, services, and coordination logic. These streams of research show that the interplay of MDE and CBSE is becoming ever more essential to address the complexity and high dynamicity of modern software systems, and their dependability as well.

3 Interplay of MDE and CBSE

MDE and CBSE can be considered as two orthogonal ways of reducing development complexity: The former shifts the focus of application development from source code to models closer to domain-specific concepts. The latter breaks down the set of desired features and their intricacy into better manageable components, from which the application can be built-up and incrementally enhanced.

When exploiting these development approaches, numerous different modelling notations and consequently several software models may be involved during the software life cycle, from requirements to specification, from analysis to code. On the one hand, effectively dealing with all the heterogeneous modelling notations that describe software systems needs to bring component-based principles at the level of the software model landscape. This is achieved by supporting, e.g. the specification of model interdependencies, and their retrieval, as well as enabling interoperability between the different notations used for specifying the software. On the other hand, MDE techniques can bring to the CBSE process the possibility to effectively reuse and integrate third-party model entities as well as to boost automation in the development process through powerful model transformations.

An effective interplay of CBSE and MDE approaches would bring benefits to both research communities. On the one hand, the research results of CBSE would benefit from the implementation and automation capabilities of MDE. This will permit to apply the best practices of CBSE to large scale systems. On the other hand, MDE would benefit from the foundational nature of CBSE approaches. Summarizing, an effective interplay of CBSE and MDE approaches could help in handling the intricacy of modern software systems, thus reducing costs and risks by: (i) enabling efficient modelling and analysis of functional as well as extra-functional properties such as, for instance, safety, reliability, availability and dependability, (ii) improving reusability through the definition and implementation of components loosely coupled into assemblies, (iii) providing automation where applicable (and favourable) in the

development process. In the last fifteen years, such cooperation has been covered by a large number of works and recognized as extremely promising; tools and frameworks have been developed for supporting this kind of integrated development process. Nevertheless, when exploiting the interplay of MDE and CBSE, clashes arise due to misalignments in the related terminology but also—and more importantly—due to differences in some of their basic assumptions and focal points.

4 In this issue

The papers in this issue address various challenges related to the combination or interplay of CBSE and MDE, ranging from verification and validation of component models to composing component behaviours, to better understanding legacy components.

In the paper "Hazard-driven realization views for Component Fault Trees" by D. S. Velasco Moncada, the author presents a modelling method and a tool solution for a hazard-centric approach for fault analysis with Component Fault Trees. This method aims to foster separation of concerns in fault modelling by leveraging hazard-specific views for Component Fault Trees. Both method and tool are applied to safety analysis in the automotive domain (direction lights functionality).

In the paper "Mixed-semantics composition of Statecharts for the component-based design of reactive systems" by B. Graics, V. Molnár, A. Vörös, I. Majzik, and D. Varró, the authors introduce the Gamma Statechart Composition Framework to facilitate the composition of semantically heterogeneous, Statechart-based software components. At the framework's core there is a modelling language describing the composition of such components from which a code generator produces composition-related Java code. The application of both is presented on an example of MDE in the railway domain.

In the paper "Interface protocol inference to aid understanding legacy software components" by K. Aslam, L. Cleophas. R. Schiffelers, and M. van den Brand, the authors present a method to infer the interface protocols of legacy software components from their behaviour models through active automata learning. In a two-step fashion, first the control-flow behaviour of a component is learned and then its interface protocol is identified through abstracting away actions related to other interfaces. The method is evaluated on an industrial codebase at ASML to automatically derive interface protocols of over 150 components.

Acknowledgements We would like to thank all authors who submitted papers as well as the reviewers for their efforts and high-quality reviews. Finally, we would like to thank Martin Schindler for his excellent support throughout the process of putting together this theme section.

Federico Ciccozzi is an Associate Professor at Mälardalen University (Sweden). His research specializes in definition of DSMLs, model transformations, system properties preservation, multi-paradigm modeling, model versioning, combination of MDE and CBSE for complex systems, blended modeling, language and compiler engineering. Federico has organized over 40 conferences, tracks, sessions, workshops and journal special issues. He has been program committee member of over 40 scientific events in the last year. He is part of the editorial board of IET Software, and guest editor of SoSyM and JISA. He has (co-)authored over 85 peer-reviewed publications. More information on him can be found at http://www.es.mdh.se/staff/266-Federico_Ciccozzi.

Antonio Cicchetti, Ph.D. is an Associate Professor at the IDT department, Mälardalen University, Sweden. He got his Ph.D. in Computer Science in 2008 at the University of L'Aquila with the thesis entitled "Difference Representation and Conflict Management in Model-Driven Engineering". His research interests include the interplay of model-driven and component-based engineering techniques and their application in the development of industrial systems. Moreover, he investigates the general problems related to the design of modelling languages, multiview systems, and model transformations, in the context of both academic research and industrial application. Further, he is interested in the concerns related to the management of evolution of both language and models. He can be reached at antonio.cicchetti@mdh.se. For more information see also http://www.es.mdh.se/staff/198-Antonio_Cicchetti.

Andreas Wortmann is a research associate at the Chair of Software Engineering of RWTH Aachen University (Germany). He received his PhD in 2016 from RWTH Aachen for his work on extensible languages for software architecture modeling. He also worked in Gothenburg (Sweden) and Rennes (France) on topics ranging from automotive software engineering to software language engineering to modeling in Industry 4.0. Additional biographical information about Andreas and details about his research interests can be found at https://www.se-rwth.de/staff/wortmann/.